DM505 Databases

Spring 2008 Project

Department of Mathematics and Computer Science
University of Southern Denmark

February 25, 2008

Introduction

The purpose of this project is to try in practice the process of creating a relational database application. This process includes design in the ER-model, transfer to the relational model, normalization of relations, implementation in a DBMS, and programming of an application providing user interaction with the database.

The project is the exam for HA(Dat) students, and part of the exam for IMADA students. Thus the project must be done individually, and *no* cooperation is allowed besides what is explicitly written in this document.[1]

PostgreSQL should be used as DBMS, and IMADA students should use Java with JDBC for programming. HA(Dat) students can use C# or PHP, providing that the result meets the specifications, and a precompiled version of the code, or alternatively a fully functional website, is provided.

System Requirements

The subject of the project is to create an electronic registration of the activities in a lunch arrangement at the university. In the lunch arrangement the participants takes turns to purchase ingredients for sandwiches, based on their sandwich consumption. The idea is to keep information about purchases made and food eaten, so as to ensure a fair shopping schedule,[2] and to generate shopping lists.

At least the following objects should be modeled in the system:

Participant Required features: Name, phone number, list of purchases[3] and a list of sandwiches eaten for each day.

Ingredient Required features: Name.

Sandwich Required features: list of ingredients, amount of each ingredient.

Current stock Required features: list of ingredients and their current amount.

Minimum inventory Required features: list of ingredients, their allowed minimum amount and their preferred amount after restocking.

(Note: the above is the specification from the user - the objects may or may not be entities in the ER-diagram you develop.)

Several minimum inventories can be envisioned to exist, for use in different situations (e.g. everyday, exam periods, Christmas time etc.).

The intended use of the system is a daily registration of sandwiches eaten for each participant. Once in a while a test can be issued that measures the current stock against a given minimum inventory. If the current stock does not meet the minimum requirements, *all* ingredients in the minimum inventory need to be restocked up to the preferred amount.

To keep the arrangement fair, each participant's total expenses should be calculated by finding the average price per sandwich,[4] and multiplying it with the total number of sandwiches eaten for that

[1] Sharing tuples for testing purposes is allowed.
[2] In practice this is done by letting the person with the most debt shop next.
[3] At least the total price of the purchase, and the date is required.
[4] Calculated by dividing the sum of all purchases with the total number of sandwiches eaten.

participant. The difference between this value and the total value of that participant's purchases is then a measure for how much the participant 'owes' to the lunch arrangement.

At least the following functionality should be provided by the application:

List all participants in the system.

List all sandwiches in the system.

List all ingredients and the amount of them.

Find purchases for a single participant, and list them by date.

Find sandwiches eaten in a given period for a participant, listed by date.

Register a sandwich as eaten by a given participant at a given date.

Write out a list of things to buy to meet a given minimum inventory.

Update the inventory by setting the amount of each ingredient to the amount preferred in a specific minimum inventory. At the same time the purchase should be registered as made by a given participant.

Get the total debt of each participant where the debt is the amount eaten for, minus the amount purchased for.

Make a list of sandwiches and the total amount that can be made with the current inventory.

A lot of update functionality that would be required in a real application has been left out, and it can be done by creating a file containing create and insert statements. This file can then be loaded into the database using `psql`. Note that this means that all functionality concerning updates and inserts can be ignored, except for the cases mentioned in the list above.

All questions are welcome, and all questions about the system requirements usually answered by the user of the system, will be answered by the lecturer.

Tasks

Your tasks are:

1. To develop an appropriate ER-model of the system.
2. To transfer the ER-model to the relational model.
3. To ensure that all relations are in BCNF form (if necessary, by decomposing relations which again should lead to a refinement of the ER-model).
4. To create these relations in a database in the PostgreSQL DBMS.
5. To program an application providing the user interaction with the system. The application should provide the functionality described above.

Input and Output

To limit the amount of programming, only a simple, command-line based interface for user interaction is required.[5] For instance, choices by the user can be input by showing a numbered list of alternatives, or by prompting for IDs of paticipants, ingredients, etc.

Test Data

The test data can be made up as you need it. A decent amount of test data must be made (at least in the order of 8 sandwiches and associated ingredients, 5 participants, and 2 minimum inventories). Sharing data with other participants in the course, to expand the amount of data, is allowed.

Formalities

In the end, a printed report of no more than 15 pages should be produced. Its main aim should be to

- Describe the design choices made during development and the reasoning behind these choices.[6]
- The structure of the final solution.

Specific items that must also be included in the report are: A diagram of your ER-model, the schemas of your relations (probably in an appendix), arguments showing that these are in BCNF form, the central parts (with explanation) of your SQL code, and a (very) short user manual for the application. The emphasis of this project is not on testing, so no *documentation* of testing is required in the report (however, your program will be tried out during grading).

The report should be produced in two stages (in the first stage, the report so far is handed in, commented on and handed back, and is extended in the last stage). Please note that the corrected version of the first part of the project *must* be handed in along with the complete report, and that its correctness will be factored in when grading.

The stages and their deadlines are:

- Task 1-3: Deadline Monday 3/3 - 2008.
- Task 4-5: Deadline Monday 17/3 - 2008.

For IMADA students

The SQL and Java code should not be given as a printed appendix, but rather be handed in using the `aflever` command on the IMADA system: Move to the directory containing your code and issue the command `aflever DM505`. This will copy the contents of the directory to a place accessible by the lecturer. Repeated use of the command is possible (later uses overwrite the contents from earlier uses). You should have a copy of your final database on the PostgreSQL

[5] You are welcome to make a more advanced user interface, but remember that this does not influence the final grade.

[6] To prepare for the writing of the report, it is a good idea to keep a log of the decisions made and of the work done.

server of the department (at the machine munch), and your code should work on this copy (i.e. it should be possible to try out your application).

For HA(Dat) students

The SQL and C#/php code should not be given as a printed appendix, but rather be made accesible to the lecturer in a digital format, e.g. by mailing it or giving an URL to a place where it can be downloaded.